

Ficha 2

Programação Imperativa

1 Arrays

1. Diga, justificando, qual o output de cada um dos seguintes excertos de código C.

(a)

```
int x [15] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};  
int *y, *z;  
y = x; z = x+3;  
for (i=0; (i<5); i++) {  
    printf ("%d_ %d_ %d\n", x[i], *y, *z);  
    y = y+1; z = z+2;  
}
```

(b)

```
void r (int a[], int n) {  
    int i, tmp;  
    tmp = a[0];  
    for (i=1; (i<n) i++)  
        a[i-1] = a[i];  
    a[n-1] = tmp;  
}  
int main () {  
    int v [10] = {1,2,3,4,5,6,7,8,9,10};  
    int i;  
    for (i=0; (i<5); i++) r (v,10);  
    for (i=0; (i<5); i++) printf ("%d_", v[i]);  
    return 0;  
}
```

(c)

```
int s (int a[], int n) {  
    int i=0;  
    i = i + a[n];  
    return i;  
}  
int main () {  
    int v [10] = {1,2,3,4,5,6,7,8,9,10};
```

```

    int i;
    for (i=0; (i<5); i++) r = s (v,i);
    printf ("%d\n", r);
    return 0;
}

```

2. Qual o resultado de, na última alínea do exercício anterior, substituir a declaração `int i=0;` por `static int i=0;`
3. Para cada uma das alíneas que se seguem, defina um programa que lê (usando a função `scanf` uma sequência de (no máximo 100) números inteiros terminada com o número 0 e imprime no ecrã:
 - (a) quantos números são superiores à média;
 - (b) a sequência sem os elementos repetidos (se os houver).
4. Defina uma função `int removeRep (int v[], int n)` que remove do *array* `v` todos os elementos repetidos (deixando apenas uma cópia). O parâmetro `n` corresponde ao número de elementos do *array* e a função deverá ainda retornar o número de elementos com que o *array* ficou.
5. Apresente uma versão mais eficiente da função da alínea anterior que assume que o array original está ordenado por ordem crescente.

2 Strings

1. Apresente uma definição de cada uma das seguintes funções (pré-definidas em C) sobre strings (arrays de caracteres terminados pelo caracter `'\0'`).
 - (a) `int strlen (char str[])` que calcula o comprimento de uma string
 - (b) `char *strcat (char s1[], char s2[])` que concatena a string `s2` a `s1` (retornando o endereço da primeira)
 - (c) `char *strcpy (char *dest, char source[])` que copia a string `source` para `dest` retornando o valor desta última.
 - (d) `int strcmp (char s1[], char s2[])` que compara (lexicograficamente) duas strings. O resultado deverá ser
 - 0 se as strings forem iguais
 - <0 se `s1 < s2`
 - >0 se `s1 > s2`
 - (e) `char *strstr (char s1[], char s2[])` que determina a posição onde a string `s2` ocorre em `s1`. A função deverá retornar `NULL` caso `s2` não ocorra em `s1`.
2. Defina uma função `void strrev (char s[])` que inverte uma string.
3. Defina uma função `void strnoV (char s[])` que retira todas as vogais de uma string.

3 Ordenação de vectores

1. Defina uma função `void insere (int v[], int N, int x)` que insere um elemento (`x`) num vector ordenado. Assuma que as `N` primeiras posições do vector estão ordenadas e que por isso, após a inserção o vector terá as primeiras `N+1` posições ordenadas.
2. Usando a função anterior, podemos definir uma função de ordenação de um vector:

```
void iSort (int v[] , int N) {  
    int i;  
    for (i=1; (i<N); i++)  
        insere (v, i, v[i]);  
}
```

Apresente uma definição alternativa deste algoritmo sem usar a função `insert`.

3. Defina uma função `int maxInd (int v[], int N)` que, dado um array com `N` inteiros, calcula o índice onde está o maior desses inteiros.
4. Use a função anterior na definição de uma função de ordenação de arrays de inteiros, que vai repetidamente calculando os maiores elementos e trocando-os com o elemento que está na última posição.
5. Apresente uma definição alternativa do algoritmo da alínea anterior sem usar a função `maxInd`.
6. Considere a seguinte definição:

```
void bubble (int v[] , int N) {  
    int i;  
    for (i=1; (i<n); i++)  
        if (v[i-1] > v[i]) swap (v, i-1, i);  
}
```

Ilustre a execução da função com um pequeno exemplo. E verifique que após terminar, o maior elemento do vector se encontra na última posição.

7. Use a função `bubble` na definição de uma função de ordenação de vectores que, repetidamente vai colocando o maior elemento do vector no fim.
8. Apresente uma definição alternativa da função pedida na alínea anterior, sem usar a função `bubble`.
9. Uma optimização frequente do algoritmo referido na última alínea, consiste em detectar se o array já está ordenado. Para isso basta que uma das passagens pelo array não efectue nenhuma troca. Nesse caso podemos concluir que o array já está ordenado. Incorpore essa optimização nas funções das alíneas anteriores.
10. Defina uma função `void merge (int r [], int a[], int b[], int na, int nb)` que, dados vectores ordenados `a` (com `na` elementos) e `b` (com `nb` elementos), preenche o array `r` (com `na+nb` elementos) com os elementos de `a` e `b` ordenados.

Usando essa função podemos construir definir o seguinte algoritmo de ordenação:

```

void mergesort (int v[] , int n, int aux[]) {
    int i , m;

    if (n>1) {
        m = n/2;
        mergesort (v , m, aux);
        mergesort (v+m, n-m, aux);
        merge (aux, v, v+m, m, n-m);
        for (i=0; (i<n); i++)
            v[i] = aux[i];
    }
}

```

Note que, se v é um vector, $v+m$ é o vector (sufixo de v) que começa na posição m , i.e.,

$$v + m = \&(v[m])$$

11. O algoritmo de quick-sort para ordenação de um vector pode ser descrito da seguinte forma:

- Começa-se por escolher um elemento do array (chamado pivot)
- De seguida, e por trocas entre elementos do vector, obtem-se uma posição p tal que o prefixo $0..p-1$ só contem elementos menores do que o pivot e o sufixo $p+1..n-1$ só contem elementos maiores ou iguais ao pivot (que se encontra na posição p)
- Aplicam-se os dois passos anteriores a ambas as partes do vector identificadas acima.

Assumindo então que existe uma função `int partition (int v[], int a, int b)` que realiza a partição do vector referida atrás, podemos codificar este algoritmo da seguinte forma:

```

void qsort (int v[] , int n){
    qsortAux (v,0,n-1);
}

void qsortAux (int v[] , int a, int b) {
    int p;
    if (a<b) {
        p = partition (v, a, b);
        qsortAux (v, a, p-1);
        qsortAux (v, p+1, b);
    }
}

```

Apresente uma possível definição da função `partition`.

4 Operações sobre Matrizes

1. Nas questões que se seguem, assuma que as dimensões das matrizes em causa são constantes (definidas à custa de uma declaração do tipo `#define`).
 - (a) Defina uma função `int triSup (float m [N] [N])` que testa se uma matriz quadrada é triangular superior, i.e., que todos os elementos abaixo da diagonal são zeros.
 - (b) Defina uma função `void transposta (float m [N] [N])` que transforma uma matriz na sua transposta.
 - (c) Defina uma função `void addTo (int a [N] [M], int b[N] [M])` que adiciona a segunda matriz à primeira.
 - (d) Defina uma função `void multM (int r [N] [M], int a [N] [K], int b[K] [M])` que coloca na matriz `r` o produto das matrizes `a` e `b`.
 - (e) Defina uma função `void pow (int r[N] [N], int a[N] [N], int e)` que coloca em `r` o resultado de multiplicar `a` por ela própria `e` vezes ($r = a^e$).
2. A família de formatos PNM (*portable anymap*) é composta por formatos textuais para representar os *pixels* de uma imagem. Na sua forma mais simples, chamada PBM (*portable bitmap*), uma imagem é representada por uma matrix de booleanos em que 0 representa branco e 1 representa preto.
 - (a) Defina uma função `void openrect (int img [N] [M], int x, int y, int h, int l)` que desenha um rectângulo de lados `l` e `h` e com vértice no ponto `(x,y)`.
 - (b) Defina uma função `void closedrect (int img [N] [M], int x, int y, int h, int l)` que desenha um rectângulo preenchido de lados `l` e `h` e com vértice no ponto `(x,y)`.
 - (c) Defina uma função que desenha uma recta que use dois pixels.
 - (d) Defina uma função que, dados o centro e o raio de uma circunferência, desenha essa circunferência.
 - (e) Defina uma função `void rotateright (int img[N] [N])` que roda uma imagem (quadrada) 90° à direita.